DOCS Design Document

Part - C

Prepared by:
Sourodeep Datta,
Tanishq Prasad,
&
Sanskar Mittal

Contents

Chapter 1 Connection Management Design

1.1 Concurrency Model

Thread Pool Architecture

- **Multi-threaded Architecture:**
  The server uses a thread-pool approach to handle multiple client connections concurrently.
- **Configurable Thread Count:**
  The number of threads is configurable, with a default of 4 threads.
- **Thread Pool Strategy:**
  Each thread runs an independent io_context event loop, which allows for efficient parallel processing of client connections and provides scalability and improved performance under high load.
- **Asio IO Context**:
  Acts as a central event dispatcher and manages asynchronous operations across threads. It distributes the client connections and I/O events.

  It lets idle threads pick up pending I/O operations, ensuring a balanced load across the thread pool.
- **Synchronization Strategy:**
  As the underlying Storage Engine supports concurrent access, no specific locking mechanisms are required for the server.

Client Session Handling

- **Session Lifecycle:**
  When a client connects, a new Session object is created. Each session manages its own socket and read/write operations.
- **Asynchronous I/O:**
  The Asio library is used non-blocking, event-driven I/O. This enables handling multiple connections without blocking threads.

Chapter 2 Protocol Implementation

The server implements a subset of the RESP-2 (Redis Serialization Protocol). At its core, the implementation supports three fundamental key-value operations: SET for storing data, GET for retrieving data, and DEL for removing entries.

The protocol parsing mechanism can handle array and bulk string types that are a part of the Redis communication protocol. The system can distinguish between different command types, parse their arguments, and ensure that each command meets the expected structural requirements.

The system has robust error-handling capabilities. When confronted with malformed or unexpected command formats, the server generates informative error responses that provide clear feedback about the nature of the parsing failure.

The server generates standardized responses that strictly adhere to the Redis protocol specifications. Each response is formatted to match the RESP-2 protocol, whether it's a successful operation, an error condition, or a data retrieval result. This ensures seamless interoperability of the server with Redis-compatible clients and the Redis-benchmark utility.